Capítulo QUINZE

Busca de Dados

Objetivos do Exame
Buscar dados usando métodos de busca das classes Stream, incluindo findFirst, findAny, anyMatch, allMatch, noneMatch.

Encontrando e Correspondendo

Buscar é uma operação comum quando se tem um conjunto de dados.

A API de Streams tem dois tipos de operação para busca.

Métodos que começam com Find:

```java
Optional<T> findAny()
Optional<T> findFirst()
```

Que buscam por um elemento em um stream. Como há a possibilidade de que um elemento não seja encontrado (se o stream estiver vazio, por exemplo), o tipo de retorno desses métodos é um Optional.

E métodos que terminam com Match:

```java
boolean allMatch(Predicate<? super T> predicate)
boolean anyMatch(Predicate<? super T> predicate)
boolean noneMatch(Predicate<? super T> predicate)
```

Que indicam se um determinado elemento corresponde ao predicado fornecido, por isso retornam um boolean.

Como todos esses métodos retornam um tipo diferente de um stream, eles são considerados operações **TERMINAIS**.

---

findAny() e findFirst()

findAny() e findFirst() praticamente fazem o mesmo, eles retornam o primeiro elemento que encontrarem em um stream:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
stream.findFirst()
    .ifPresent(System.out::println); // 1

IntStream stream2 = IntStream.of(1, 2, 3, 4, 5, 6, 7);
stream2.findAny()
    .ifPresent(System.out::println); // 1
```

Se o stream estiver vazio, eles retornam um Optional vazio:

```java
Stream<String> stream = Stream.empty();
System.out.println(
    stream.findAny().isPresent()
); // false
```

Claro, você pode combinar esses métodos com outras operações de stream:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
stream
    .filter(i -> i > 4)
    .findFirst()
    .ifPresent(System.out::println); // 5
```

Quando usar findAny() e quando usar findFirst()?
Ao trabalhar com streams paralelos, é mais difícil encontrar o primeiro elemento. Neste caso, é melhor usar findAny() se você não se importar com qual elemento será retornado.

---

anyMatch(), allMatch() e noneMatch()

anyMatch() retorna true se qualquer um dos elementos de um stream corresponder ao predicado fornecido:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream.anyMatch(i -> i%3 == 0)
); // true
```

Se o stream estiver vazio ou se não houver nenhum elemento correspondente, este método retorna false:

```java
IntStream stream = IntStream.empty();
System.out.println(
    stream.anyMatch(i -> i%3 == 0)
); // false

IntStream stream2 = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream2.anyMatch(i -> i%10 == 0)
); // false
```

allMatch() retorna true apenas se TODOS os elementos do stream corresponderem ao predicado fornecido:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream.allMatch(i -> i > 0)
); // true

IntStream stream2 = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream2.allMatch(i -> i%3 == 0)
); // false
```

Se o stream estiver vazio, este método retorna TRUE sem avaliar o predicado:

```java
IntStream stream = IntStream.empty();
System.out.println(
    stream.allMatch(i -> i%3 == 0)
); // true
```

noneMatch() é o oposto de allMatch(), retorna true se NENHUM dos elementos do stream corresponder ao predicado fornecido:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream.noneMatch(i -> i > 0)
); // false

IntStream stream2 = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream2.noneMatch(i -> i%3 == 0)
); // false

IntStream stream3 = IntStream.of(1, 2, 3, 4, 5, 6, 7);
System.out.println(
    stream3.noneMatch(i -> i > 10)
); // true
```

Se o stream estiver vazio, este método também retorna TRUE sem avaliar o predicado:

```java
IntStream stream = IntStream.empty();
System.out.println(
    stream.noneMatch(i -> i%3 == 0)
); // true
```

Avaliação Curta (Short-circuiting)

Todas essas operações usam algo semelhante à avaliação curta dos operadores && e ||.

Avaliação curta significa que a avaliação para assim que um resultado é encontrado.

No caso das operações find*, é óbvio que elas param no primeiro elemento encontrado.

Mas no caso das operações *Match, pense bem: por que você avaliaria todos os elementos de um stream quando, ao avaliar o terceiro elemento (por exemplo), já pode saber se todos ou nenhum dos elementos correspondem?

Considere este código:

```java
IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
boolean b = stream
    .filter(i -> {
        System.out.println("Filter:" + i);
        return i % 2 == 0; })
    .allMatch(i -> {
        System.out.println("AllMatch:" + i);
        return i < 3;
    });
System.out.println(b);
```

Qual seria a saída?

Saída:

Como pode ver, antes de tudo, operações em um stream não são avaliadas sequencialmente (neste caso, primeiro filtrar todos os elementos e depois avaliar se todos os elementos correspondem ao predicado do allMatch()).

Segundo, podemos ver que assim que um elemento passa no predicado do filtro (como 2), o predicado do allMatch() é avaliado.

Finalmente, podemos ver a **avaliação curta** em ação. Assim que o predicado do allMatch() encontra um elemento que não retorna true (como 4), as duas operações de stream são canceladas, nenhum outro elemento é processado e o resultado é retornado.

Apenas lembre-se:

- Com algumas operações, o stream inteiro não precisa ser processado.
- Operações em stream não são realizadas de forma sequencial.

Pontos-Chave

- A API de Streams tem dois tipos de operação para busca:
    - Métodos que começam com Find:
      Optional<T> findAny()
      Optional<T> findFirst()
    - Métodos que terminam com Match:
      boolean allMatch(Predicate<? super T> predicate)
      boolean anyMatch(Predicate<? super T> predicate)
      boolean noneMatch(Predicate<? super T> predicate)
- Ambos os tipos são considerados **operações TERMINAIS**.
- findAny() e findFirst() praticamente fazem o mesmo, retornam o primeiro elemento que encontrarem em um stream. Se o stream estiver vazio, retornam um Optional vazio.
- Ao trabalhar com streams paralelos, é mais difícil encontrar o primeiro elemento. Neste caso, é melhor usar findAny() se você não se importar com qual elemento será retornado.
- anyMatch() retorna true se algum elemento no stream corresponder ao predicado fornecido. Se o stream estiver vazio ou nenhum elemento corresponder, retorna false.
- allMatch() retorna true somente se **TODOS** os elementos no stream corresponderem ao predicado fornecido.
- noneMatch() retorna true se **NENHUM** dos elementos no stream corresponder ao predicado fornecido.
- Tanto allMatch() quanto noneMatch() retornam true se o stream estiver vazio.
- Todas essas operações são **de avaliação curta**, ou seja, a avaliação para assim que um resultado for encontrado.

Autoavaliação

1. Dado:

```java
public class Question_15_1 {
    public static void main(String[] args) {
        Stream<Integer> s = Stream.of(100, 45, 98, 33);
        s.anyMatch(i -> i > 50)
            .findAny()
            .ifPresent(System.out::println);
    }
}
```

Qual é o resultado?

A. 100
B. 98
C. Nada é impresso
D. A compilação falha

---

2. Qual dos seguintes métodos da interface Stream retorna um tipo Optional?

A. filter()
B. findMatch()
C. findAny()
D. anyMatch()

---

3. Dado:

```java
public class Question_15_3 {
    public static void main(String[] args) {
        IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
        stream.allMatch(i -> {
            System.out.print(i);
            return i % 3 == 0;
        });
    }
}
```

Qual é o resultado?

A. 1234567
B. 36
C. 1
D. A compilação falha

---

4. Dado:

```java
public class Question_15_4 {
    public static void main(String[] args) {
        IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7);
        stream.filter(i -> {
                return i > 3;
        }).anyMatch(i -> {
                System.out.print(i);
                return i % 2 == 1;
        });
    }
}
```

Qual é o resultado?

A. 45
B. 5
C. 4567
D. A compilação falha